CS-E4190

Cloud Software and Systems

Microservice patterns

Matti Siekkinen

Based on slides
by Mario Di Francesco
For classroom use only,
no unauthorized distribution

Microservice patterns

Communication styles

- synchronous: wait for completion
- asynchronous: provide a callback for later execution

Workflows

- collaborating microservices
- primarily to implement a business process
- example: fulfilling order in online store
  - check if item is in stock, reserve for order in warehouse
  - process payment and award loyalty points
  - retrieve and package good from warehouse, ship it to customer

Source: Sam Newman, "Building Microservices", 2nd edition, O'Reilly Media, 2021

Distributed transactions

Transaction

- a set of one or more actions on a resource, treated as a single unit
  - it succeeds only if all operations therein succeed, otherwise it fails
- commonly used in databases
  - ACID properties: atomicity, consistency, isolation, durability

Two-phase commit

- algorithm for transactional changes in a distributed system
  - voting phase: coordinator asks workers involved in a transaction if a status change can be made
  - commit phase: changes are made if workers replied affirmatively
- subject to different types of failures, slow with many participants

Source: Sam Newman, "Building Microservices", 2nd edition, O'Reilly Media, 2021 ■ Chapter 6

Sagas

What is a saga?

- approach to coordinate multiple changes in state without locking resources for long time
- approach: break down long-lived transactions into small independent pieces

Recovering from failures

- backward: revert failure and cleanup afterwards (i.e., roll-back)
- forward: keep processing and retry failed trasactions as needed

Realizing sagas

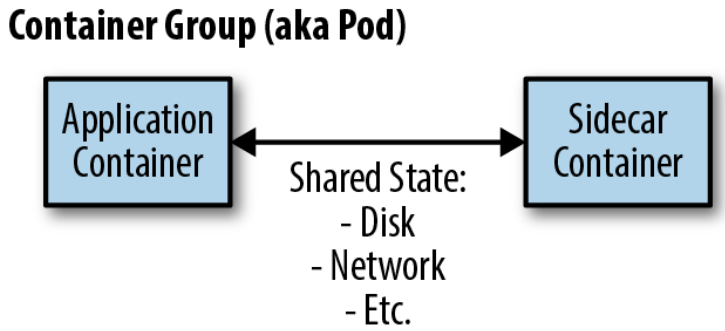
- orchestration: coordinator controls execution and triggers roll-back
- choreography: distributes responsibility to multiple services

Source: Sam Newman, "Building Microservices", O'Reilly Media, 2021 ■ Chapter 6

Sidecar

Sidecar pattern

- a single-node pattern consisting of two containers
  - (pre-existing) application container
  - sidecar container, transparently adding functionality



Sidecar container

- on the same container group with the application container
  - a pod in Kubernetes
  - shared resources and lifecycle

Source: Brendan Burns, "Designing Distributed Systems", O'Reilly Media, 2018 ■ Chapter 2

Service mesh

Overview

- requires little or no changes to application code
- programmable framework
  to observe, secure, and connect microservices
  - control plane for configuration
  - data plane with client-side proxies deployed as sidecars

Typical use cases

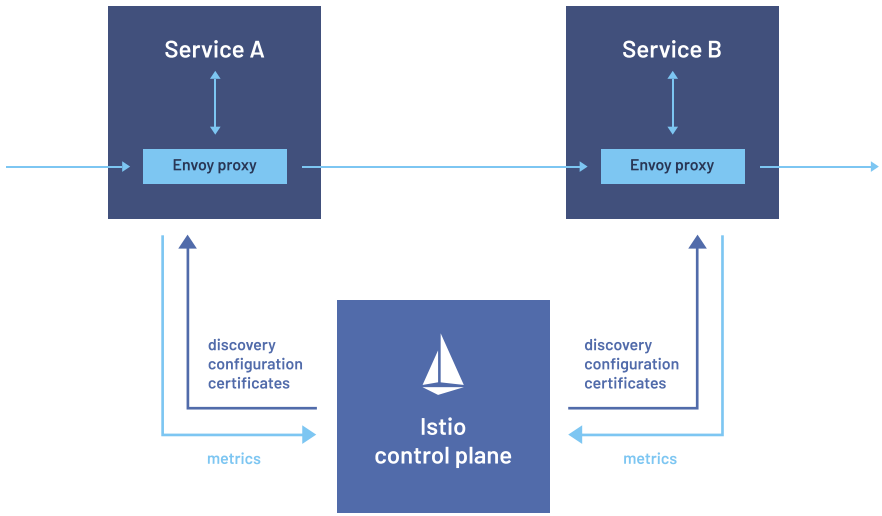
- observability: connectivity analysis, distributed tracing
- security: access control, auditing, transparent TLS encryption
- network management: routing rules, traffic splitting

Source: Lin Sun and Daniel Berg, "Istio Explained", O'Reilly Media, 2020 ■ Chapter 1

Use case: Istio

Istio

- most widely-used open-source service mesh, specifically targeted for Kubernetes
- uses Envoy proxies as sidecars



Sources: The Istio Service mesh; Lin Sun and Daniel Berg, "Istio Explained", O'Reilly Media, 2020 ■ Chapter 1